I.J. Intelligent Systems and Applications, **2017, 1, 75-84**
Published Online January 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2017.01.08

High Performance Computation of Big Data: Performance Optimization Approach towards a Parallel Frequent Item Set Mining Algorithm for Transaction Data based on Hadoop MapReduce Framework

Guru Prasad M S
SDMIT/CSE, Ujire, 577240, India
E-mail: guru0927@gmail.com

Nagesh H R and Swathi Prabhu
MITE/CSE, Moodbidri, 574227, India
SMVITM/CSE, Udupi, 576115, India
E-mail: nageshhrcs@reddifmail.com, prabhuswathi2@gmail.com

***Abstract*—**The Huge amount of Big Data is constantly arriving with the rapid development of business organizations and they are interested in extracting knowledgeable information from collected data. Frequent item mining of Big Data helps with business decision and to provide high quality service. The result of traditional frequent item set mining algorithm on Big Data is not an effective way which leads to high computation time. An Apache Hadoop MapReduce is the most popular data intensive distributed computing framework for large scale data applications such as data mining. In this paper, the author identifies the factors affecting on the performance of frequent item mining algorithm based on Hadoop MapReduce technology and proposed an approach for optimizing the performance of large scale frequent item set mining. The Experiments result shows the potential of the proposed approach. Performance is significantly optimized for large scale data mining in MapReduce technique. The author believes that it has a valuable contribution in the high performance computing of Big Data.

***Index Terms*—**Big Data, Hadoop, MapReduce, Hadoop Distributed File System (HDFS), Apriori MapReduce, FP-growth MapReduce.

I. INTRODUCTION

We live in the Big Data Era. Big Data is a broad term that describes a massive volume of structured, semi-structured and unstructured data. Due to the advent of new technologies and digital world of data is expanded to 10 zettabytes (10^{21} bytes). Huge amount of data is generated from social networking sites, e-commerce, on-line banking, weather stations, market transactions etc. Big Data is mainly characterized by 3 V’s extreme volumes, extreme variety and extreme velocity. Volume can vary beyond zettabytes. Velocity defines the speed at which data is generated and huge variety of data can be used. It is really critical to business enterprises and its emerging as one of the most important technologies in modern world. Many business enterprises accumulate large quantities of data from customer transactions; they handle more than one billion customer transaction every day. For example, EBay has 50 petabytes of data it captures 50 terabytes every day, US retailers have around 500 petabytes of data, Amazon is world's biggest retail store which has billions of active customer data etc. Huge amount of data continuously collected and stored in their data warehouse. Now business organizations are interested in extracting knowledgeable information from stored data. The information contained in transaction database is large. So it is very difficult to understand and also difficult to extract knowledgeable information from this huge dataset. To solve this problem the technique called frequent item mining is used. This technique finds the frequency of item purchased together. It is useful in extracting hidden predictive information from large data sets. And it is a powerful technology with extreme potential to help organizations. It focus on the most important information in their data warehouse. The frequent item mining technique predicts future trends and behaviors. It allows businesses to undertake proactive, knowledge-driven decisions. Apriori and FP-growth are the most famous algorithms to discover frequent patterns in large data sets. However, the existing data mining tools based on sequential Apriori and FP-growth algorithms are not efficient to mine a huge transaction data.

We would require a robust distributed computing infrastructure that can store, manage and process huge amounts of data in short time. It can protect data security and privacy. Doug Cutting, Mike Cafarella and their team took the challenge and designed new powerful distributed computing open source project called Hadoop. Today Hadoop is the most popular and powerful framework with the MapReduce programming model for distributed analysis of Big Data. The main features of Hadoop are, It can be easily accessible, robust in nature and scalable. It allows to store, manage and process Big Data in a distributed environment across a cluster of computers that works with simple parallel programming model. It is designed to easily scale up from single machine to thousands of machines. The main power of Hadoop is Hadoop Distributed File System (HDFS) and MapReduce programming model. Hadoop can be compatible with any of the mountable distributed file systems such as Hadoop Distributed File System that can operate on existing hardware. HDFS has a very good durability and uses cheap hardware. It is highly fault tolerant distributed file system, secures data stored on the clusters. The HDFS is based on the Google File System and provides a Distributed File System (DFS) that is designed to run on a large cluster (thousands of computers) which handles huge amount of data such as terabytes, petabytes, etc. The HDFS runs across the nodes in a Hadoop cluster which connects the file systems together with huge input and output data that make them into one big file system. Hadoop running MapReduce programs written in various languages like Java, Python and C++. It is a programming model for easy writing applications which processes huge amounts of data in-parallel on a large cluster of hardware's in a reliable, robust, fault-tolerant manner. Hadoop provides its own set of basic types that are optimized for network serialization. Jobs are divided into two tasks, such as map tasks and reduce tasks. Many works were proposed on MapReduce based Apriori and FP-growth algorithm to mine frequent items over large transaction data. However, the performance of frequent item mining algorithms on Hadoop MapReduce framework is not optimal; it takes more HDFS disk space and not so effective. Our Experiment results show that, the proposed novel approach effectively reduces the HDFS disk space utilization and minimizes the execution time.

The major contributions of this paper are as follows:

1. Minimizes the execution time of frequent item mining on Hadoop MapReduce framework.
2. Effectively reduce the HDFS disk space utilization of the DataNode to store transaction data.
3. Potential of the proposed method is demonstrated.
4. Provided valuable contribution for high performance computation of Big Data.

The rest of the paper is organized as follows. Section II is an illustration of related works about the proposed topic. Section III discus the Backgrounds of the proposed work. Section IV proposes the methodology of optimizing the performance of parallel frequent mining algorithm. Section V presents comprehensive experiment results. Section VI concludes the paper.

II. Related Work

Hui Chen et al [1] described about Big Data size. It has increased from terabytes to petabytes. Sequential data mining algorithm does not satisfy the needs of big data mining. In this paper, the author designed a novel parallel algorithm for mining frequent item sets information over big transaction data based on Hadoop MapReduce Framework. The simulation result proves that, the proposed parallel algorithm is efficient and scalable, and can be used to efficiently mine frequent patterns in big data.

Zahra Farzanyar et al [2] illustrates mining information produced in the social network environments can be extremely useful. But, with the rapid growth of social network data towards a terabyte or more. Most of the traditional mining algorithms become ineffective due to either too huge resource requirement or too much communication. In this paper, the author proposed an efficient frequent item set mining algorithm called IMRApriori based on a MapReduce framework which deals with Hadoop cloud which is a parallel store and computing platform. Author work demonstrates experimental results to corroborate the theoretical claims.

Yanfeng Zhang et al [3] says new stream of data and updates are arriving every minute. The results of data mining applications become slow and consume more time. Incremental processing was a promise method to refreshing mining results; it uses previously saved results to avoid the expense of re-computation from the start. In this paper the author proposed i^2 MapReduce, a novel incremental process extension to MapReduce which is a most widely used framework for Big Data computing. Experimental results, shows significant performance enhancement of i^2 MapReduce compared to both plain and iterative MapReduce performing re-computation.

LI Bing et al [4] described data mining algorithm that can process massive amounts of data have recently been referred to as big data algorithms. In this paper the author proposed a big data algorithm to extract knowledgeable information from twitter data. Furthermore the proposed algorithm is parallelizing based on the most promising MapReduce framework. By the massive experiments on large twitter data, the potential of the proposed big data algorithm was demonstrated. Computationally, the speed of data processing was increased significantly, despite of increases in data set size. Results showed that the acceleration ratio increases as the data size increases and as the number of DataNodes increases.

Sheela Gole et al [5] illustrates the enormous volume of data is getting explored through Internet of Things from a variety of sources such as e-commerce, mobiles , sensors, social media, internet applications, called as Big Data. Big Data cannot be handled by traditional tools and techniques. The Big Data mining is more necessary in order to extract knowledgeable value from large amounts of data which could give better insights using efficient methods. To overcome the existing data mining algorithm limitation, MapReduce is used for parallel processing of Big Data, having features such as high scalability, fault-tolerance and robustness which help to handle the problem of giant datasets. In this paper the author proposed novel method ClustBigFIM, extended BigFIM algorithm works on MapReduce framework for mining meaningful information from giant datasets.

Yen-hui Liang et al [6] described frequent item set mining. It is a significant research topic because it is extensively adapted in real world to find the frequent item sets and to mine human behavior patterns. The limitation of classical frequent item set mining process is high time and memory consumptions. In this paper the author proposed a novel distributed frequent item set mining algorithm called Sequence-Growth, and implement it based on MapReduce framework. Proposed algorithm applies the concept of lexicographical order to create a tree called "lexicographical sequence tree" which allows to find all frequent item sets without repeated search over the transaction datasets. To test the performances of novel method, author conducted varied aspects of experiments on MapReduce framework with large datasets. The experiment results showed the good suitability of Sequence-Growth to mine frequent item set.

Zhuobo Rong et al [7] explores the problem of existing sequential Apriori and FP-growth algorithm on the single machine environment are high memory consumption and low performance computing . In this paper, author proposed parallel environment for frequent item mining algorithm, implemented based on MapReduce framework. The experimental results showed better efficiency and scalability of proposed parallel Apriori and FP-growth algorithm.

III. Backgrounds

A. Hadoop

Hadoop is a powerful open-source, Java based framework that is managed by apache software which offers a robust platform to work with big data. It was inspired by Google's MapReduce, a software framework in which an input data are broken down into numerous equal size data parts. Any of these parts can run on any node in the cluster. It is now a top level Apache project being built and is used by the community of contributors from all over the world. Hadoop is not a replacement for database, data warehouse or ETL (Extract, Transform and Load) strategy. It supports the computation of huge data sets in a distributed computing environment.

Hadoop is a Master/Slave architecture cluster; consist of one Master Node named as NameNode and n number of Slave nodes named as DataNode. It allows to store, secure and process Big Data in a distributed environment across clusters of computers that works with MapReduce parallel programming model. It is designed to scale up from single machine to thousands of machines. It can be compatible with any of the mountable distributed file systems such as Hadoop Distributed File System(HDFS). HDFS has a very good durability and can use cheap hardware. It is highly fault tolerant distributed file system that is responsible for storing data on the clusters. The HDFS is based on the Google File System and provides a distributed file system to that handle huge amount of data such as terabytes, petabytes, etc.

B. MapReduce

MapReduce is a parallel programming model and powerful interface first developed by Google using which we can write applications to process a big amount of data on large clusters of commodity hardware in a reliable manner. A MapReduce job splits a huge data set into equal size of independent chunks and organizes them into key, value pairs for parallel processing. The speed and reliability of the cluster are improved by parallel processing and solutions obtain more quickly and with greater reliability.

The MapReduce algorithm consists of two tasks namely map and reduce. The map function is to process the input file. Map function reads input data line by line and then it will process the data and creates several chunks of data. The reduce function takes the output of the mapper as input and process the data. After processing is completed it produces a new set of output, which will be stored in HDFS. As the name indicates MapReduce, the reduce task is always performed after the map job. During a MapReduce job, Hadoop sends the Map and Reduce task to the appropriate DataNodes in the cluster. All the details of data passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes are managed by MasterNode. After completion of the given jobs the MasterNode collects result from all DataNodes.

The MapReduce framework operates solely on <key, value> pairs, i.e. the framework gives <key, value> pairs as input to the job and produces the output of the job as a set of <key, value>pairs of different types.

Input and Output types of a MapReduce job are as follows:

(input) <k1, v1> → map ()→ list (<k2, v2>) (output)
(input) <k2, list (v2)> → reduce () → list<k3, v3> (output)

C. Association Analysis

Association analysis is an approach for discovering interesting relationships between items in large transaction. It is intended to identify strong rules for discovering interesting relationships hidden in large data sets. Based on the idea of strong rules, Rakesh Agrawal et.al .introduced association rules for discovering regularities between items in large-scale grocery transaction. For example , the rule {Bread, Eggs} → {milk} found in the sales data of a grocery stores would indicate that, if a customer buys Bread and Eggs together, they are likely to buy Milk. Such information helps retailers to identify which items are purchased together.

An association rule is an expression of the type $X \rightarrow Y$, where X and Y are disjoint item sets i.e., $X \cap Y = Ø$. The strength of an association rule will be measured in terms of its support and confidence. Support determines how typically a rule is applicable to given information set while confidence determines however oftentimes things in Y seem in transactions that contain X. The formal definitions of these metrics are

Support (X U Y)=(number of transaction containing in both X and Y)/(total number of transaction)

Confidence (X|Y)=(number of transaction containing in both X and Y) / (number of transactions containing X)

The association rule mining from large databases is a two-step process.

1. Find all the frequent item set, that satisfy the minimum support count threshold, itemset >= minimum support count.
2. Generate strong association rules satisfying minimum support and confidence.

Efficient algorithm to generate frequent itemsets and association rule are Apriori and Fpgrowth.

D. Apriori

Apriori is an efficient algorithm proposed by R. Agrawal and R. Srikant in 1994 for frequent item set mining and association rule learning. It is designed to operate on databases with a large number of transactions. It applies an iterative approach to find the frequent item set that satisfies the minimum support count threshold. The pseudo code of apriori is shown below.

Apriori Pseudo Code

```
Algorithm 1: Apriori
Input:
DB, (Database of transactions),
min_sup (minimum support count threshold),
Output:
L, frequent itemsets in D.
Method:
Begin
1. $F_1$ =find_frequent_1-itemsets(DB)
2. For (i=2;$L_{i-1}$!=ϕ;i++) {
3. $C_k$=apriori_generation($F_{k-1}$,min_sup);
4. For each transaction t ∈ DB;
5. $C_t$=subsets($C_k$,t);
6. For each candidate C ∈ $C_t$
7. c.count++;
8. End For each
9. End For each
10. $F_K$={C ∈ $C_k$| c.count>=min_sup }
11. End For
12. Return F U $F_K$
End
```

E. FP-growth

FP-growth, proposed by Jiawei Han, Jian Pie and Yiwen Yin [ ], is an efficient and scalable algorithm to find frequent item sets. It uses novel data structure called frequent pattern tree (FP-tree) which stores information about frequent patterns.

Table 1. Transaction Database

TID	Items
T1	{I1,I2}
T2	{I2,I3,I4}
T3	{I1,I2,I3}
T4	{I1,I3}
T5	{I1,I4}

Steps to construct FP-tree are as follows:

1. Initially create the root node of the FP-tree, label it as null symbol.
2. Scan the transaction databasen calculate support count of each item and arrange items in decreasing support count. For the transaction database shown in Table 1, I1 is the most frequent item followed by I2, I3 and I4.
3. Read the first transaction {I1, I2}, construct the first branch of tree with two nodes, (I1,1) and (I2,1). A branch is then formed from *null→I1→I2.*
4. Read the second transaction {I2,I3,I4}, construct the second branch of tree with three nodes, (I2,1) ,(I3,1) and (I4,1). A branch is then formed from *null→I2→I3→I4.*
5. Repeat the process same for all transaction in database. The resulting FP tree for Table1 is shown in Fig 1.

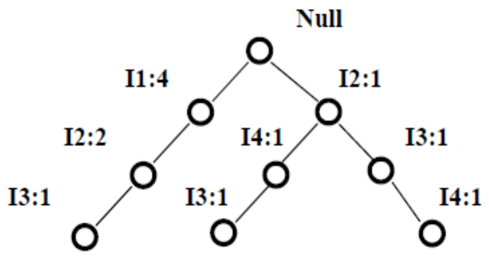

Fig.1. FP-Tree

IV. METHODOLOGY

To analyse frequent items we develop a novel methodology as shown in Fig 2.The detailed description of the methodology is as follows:

A. Data Pre-processing

Data pre-processing is an important step in the proposed methodology. It is a data mining technique that involves transforming transaction raw data into an understandable format that will be more easily and effectively processed by the Hadoop. Data pre-processing is categorized into data cleaning, data integration, data transformation, data reduction and data discretization. In the proposed methodology data cleaning technique is used to remove unwanted data from the transaction data.

B. Data Conversion

Data conversion is a proposed approach to convert the grocery item's name into unique_ID

Pseudo code of data conversion is as follows:

```
step 1. Read the transaction data.
step 2. Calculate n (number of items).
step 3. for 0 to n-1
        assign unique_ID to each item.
step 4. Write item name and unique_ID into file1.
step 5. Using file1 replace each item in transaction data by unique_ID.
step 6. Write output of step5 into file2.
```

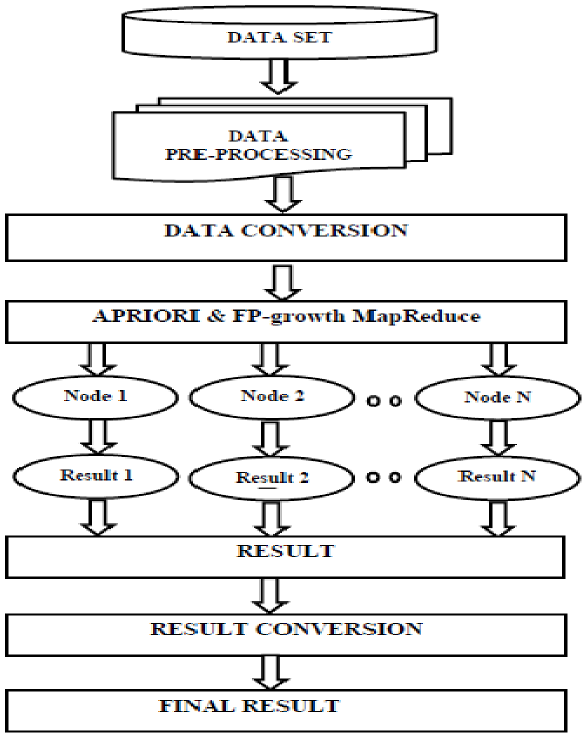

Fig.2. Proposed Methodoloy

C. Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is a Java based distributed file system designed to run on a thousands of computer. It is efficient and reliable data storage layer of Hadoop. It is popular because of robustness, data organization, accessibility and data replication. It has Master/Slave architecture, consist of one Master node (Name node) and N number of Slave node (Data node).

HDFS works as follows:

1. It divides large volume of transaction data into M blocks of equal size (D_1, D2, D3,…….., DM)
2. It replicates data blocks (D_1, D_2, D_3,…….., D_M) and stores data blocks in Data nodes.

D. Frequent item mining algorithms based on MapReduce

This section shows MapReduce based implementation of Apriori, FP-growth and Association rule mining algorithms.

Implementation of Apriori algorithm based on MapReduce.

```
Algorithm 2: Apriori in MapReduce
Apriori_Mapper<K1,V1,K2,V2>
1. Mapper input<K1,V1>=map(longwritable key, text value);
2. string transaction=value.toString();
3. List<string> itemsets = getitemsets(transaction.split(","));
4. for(string itemset:itemsets)
5. itemset.set(itemset.replaceall(",");
6. Mapper output<K2,V2>=output.collect(itemset, new intwritable(1));

Apriori_Reducer<K2,V2,K3,V3>
7.Reducer input<K2,V2>=reduce(Text key, iterator<intwritable> values);
8. sum=0;
9. while(values.hasNext())
10. sum+=values.next().get();
11. if(sum>=minimum_support_count)
12. Reducer output<K3,V3>=output.collect(key, new text(Integer.toString(sum)));
```

Implementation approach

Implementation steps of Apriori in MapReduce are as follows:

Step 1: Input to the Mapper is <K1,V1>, where K1 is the data blocks name and V1 is itemsets inside the file.

Step 2: Mapper scans the itemsets and generates all subsets from the itemsets.

Step 3: Output of Mapper is <K2,V2>, where K2 is the subsets of itemset and V2 is the value of each subset.

Step 4: Reducer reads the output of Mapper<K2,V2> and adds all subsets of itemset.

Step 5: If sum >= minimum_support_count, then the corresponding item subsets is considered.

Step 6: Output of Reducer is <K3,V3>, where K3 is the itemsets and V3 is the count of itemsets.

Implementation of FP-Growth algorithm based on MapReduce.

```
Algorithm 3: FP-Growth in MapReduce
FP-Growth_Mapper<K1,V1,K2,V2>
1. Mapper input<K1,V1>=map(longwritable key, text value)
2. String[] items = splitter.split(input.toString());
3. OpenIntHashSet itemSet = new OpenIntHashSet();
4. for (String item : items) {
5. if (fMap.containsKey(item) && !item.trim().isEmpty())
{
6. itemSet.add(fMap.get(item));}}
7. IntArrayList itemArr = new
IntArrayList(itemSet.size());
8. itemSet.keys(itemArr);
9. itemArr.sort();
10. OpenIntHashSet groups = new OpenIntHashSet();
11. for (int j = itemArr.size() - 1; j >= 0; j--) {
12. int item = itemArr.get(j);
13. int groupID = ARM.getGroup(item, maxPerGroup);
14. if (!groups.contains(groupID)) {
15. IntArrayList tempItems = new IntArrayList(j + 1);
16. tempItems.addAllOfFromTo(itemArr, 0, j);
17. context.setStatus("Parallel FPGrowth: Generating
Group Dependent transactions for: "+ item);
18. wGroupID.set(groupID);
19. context.write(wGroupID, new
TransactionTree(tempItems, 1L));}
20. groups.add(groupID)}}
21. protected void setup(Context context) throws
IOException,InterruptedException {
22. super.setup(context);
23. int i = 0;
24. for (Pair<String, Long> e :
ARM.readFList(context.getConfiguration())) {
25. fMap.put(e.getFirst(), i++);}
26. Parameters params=
newParameters(context.getConfiguration().get(ARM.PFP
_PARAMETERS,""));
27. splitter =
Pattern.compile(params.get(ARM.SPLIT_PATTERN,AR
M.SPLITTER.toString()));
28. maxPerGroup =
params.getInt(ARM.MAX_PER_GROUP, 0);}}
29. Mapper output<K2,V2>=output.collect(intwritable,
Transactiontree);
```

FP-Growth_Reducer<K2,V2,K3,V3>

```
30. Reducer input<K2,V2>=reduce(IntWritable key,
Iterable<TransactionTree> values);
31. TransactionTree cTree = new TransactionTree();
32. for (TransactionTree tr : values) {
33. for (Pair<IntArrayList, Long> p : tr) {
34. cTree.addPattern(p.getFirst(), p.getSecond());}}
35. List<Pair<Integer, Long>> localFList =
Lists.newArrayList();
36. for (Entry<Integer,
org.apache.commons.lang3.mutable.MutableLong>
fItem : cTree.generateFList().entrySet()) {
37. localFList.add(new Pair<Integer,
Long>(fItem.getKey(), fItem.getValue().toLong()));}
38. Collections.sort(localFList,new
CountDescendingPairComparator<Integer, Long>());
39.FPGrowth<Integer> fpGrowth = new
FPGrowth<Integer>();
40. fpGrowth.generateTopKFrequentPatterns(new
IteratorAdapter(cTree.iterator()),localFList,
minSupport,maxHeapSize,new
HashSet<Integer>(ARM.getGroupMembers(key.get(),
maxPerGroup, numFeatures).toList()),new
IntegerStringOutputConverter(
new  ContextWriteOutputCollector<IntWritable,
TransactionTree, Text, TopKStringPatterns>(
context), featureReverseMap),new
ContextStatusUpdater<IntWritable, TransactionTree,
Text, TopKStringPatterns>(context));}@Override
41. protected void setup(Context context) throws
IOException,InterruptedException {
42. super.setup(context);
43.Parameters params = newParameters(
context.getConfiguration().get(ARM.PFP_PARAMETER
S, ""));
44. for (Pair<String, Long> e :
ARM.readFList(context.getConfiguration())) {
45. if (!e.equals("dataset")) {
46.featureReverseMap.add(e.getFirst());
47. freqList.add(e.getSecond());}}
48. maxHeapSize =
Integer.valueOf(params.get(ARM.MAX_HEAPSIZE,
"50"));
49. minSupport =
Integer.valueOf(params.get(ARM.MIN_SUPPORT, "5"));
50. log.info("Support count: " + minSupport);
51. maxPerGroup =
params.getInt(ARM.MAX_PER_GROUP, 0);
52. numFeatures = featureReverseMap.size();}}
53. Reducer output<K3,V3>=output.collect(key,
TopKStringPatterns);
```

Implementation approach

Implementation steps of FP-Growth in MapReduce are as follows:

Step 1: Input to the Mapper is <K1,V1>, where K1 is the data id and V1 is itemsets inside the file.

Step 2: Mapper scans the itemsets and generates FP-tree

Step 3: Output of Mapper is <K2,V2>, where K2 is the groupid and V2 is the transaction tree.

Step 4: Reducer reads the output of Mapper<K2,V2> and finds the frquency of items

Step 5: If frequency>= minSupportt, then the corresponding transaction is considered.

Step 6: Output of Reducer is <K3,V3>, where K3 is the key and V3 is TopKStringPatterns.

Implementation of Association analysis algorithm based on MapReduce

Algorithm 4: Association analysis in MapReduce
Association_Mapper<K1,V1,K2,V2>

```
1.Mapper input<K1,V1>=Map(Longwritable key, Text
value);
2.string valuesplit=value.toString().split("\t");
3.List<string>subitemsets=getitemsets(items);
4.for(string itemset:subitemsets)
5.itemset.set(itemset.replaceall(" ",""));
6.value.set(valuesplit[0]+";"+valuesplit[1]);
```

```
7.Mapper output<K2,V2>=output.collect(itemset,val);

Association_Reducer<K2,V2,K3,V3>
8.Reducer input<K2,V2>=Reduce(Text key,
Iterator<Text> values);
9.Hashing<string,integer> hashing=new hashing<string,
integer>();
10.while(values.hasNext())
11.string val=values.next().tostring;
12.Hashing.put(key+"->"+getseparateitem)
13.Iterator<string> iterator=hashing.keyset().iterator();
14. while(iterator.hasNext())
15. string k=iterator.next().tostring();
16. int v=hashing.get(k);
17. Reducer output<K3,V3>=output.collect(new Text(k),
new FloatWWritable(v/keycount));
```

E. Result Conversion

Result conversion is a proposed approach to convert the frequent unique_ID into item's name

Pseudo code of result conversion is as follows:

```
Step 1. Read the result of frequent unique_ID.
Step 2. Using file1 replace each unique_ID in result file by item's name.
Step 3. Write frequent item's name into file3.
```

V. Results and Discussion

The Performance of the Hadoop MapReduce framework with respect to the time taken to move files from local file system to the Hadoop Distributed File System, memory usage of the HDFS to store data sets and the execution time of frequent items and association analysis was initially benchmarked with the baseline Hadoop (traditional) and then compared with results obtained using the Optimize Hadoop (proposed).

A. Experimental Environment

For the performance evaluation we considered Hadoop five nodes cluster with homogeneous hardware property, i.e. Each node in the cluster has a 3.8 GB RAM, Intel® Core i5 3470 CPU @3.20GHz * 4 processor. We setup cluster on Ubuntu 15.04 with Hadoop 1.7.2 stable release used oracle jdk 1.8 and ssh configuration to manage Hadoop daemons. Our cluster setup is having 1 NameNode and 5 DataNodes for the purpose of an experiment. Configuration files such as mapred-site.xml, core-site.xml, hdfs-site.xml, yarn-site.xml are setup by default values with replication factor 2 and data block size 64 MB.

B. Performance Measurement Parameters

The performance of the Hadoop MapReduce framework was measured on the following parameters.

1. Analysis of Time taken to move files from local file system to HDFS
2. Memory usage of the DataNode to store data blocks.
3. Time taken by the MapReduce to mine frequent items and association analysis.

C. Experiments

1) Analysis of Time taken to move files from local file system to HDFS

Files move operation is performed on both the traditional method and the proposed method. We recorded time taken to move files from local file system to Hadoop Distributed File System. In the traditional method where we directly moved entire input file of size 10 GB from local file system to HDFS. In the proposed method we converted the transaction item's name into unique ID, thereby reducing transaction data set size of 10 GB to 1.5 GB. Then data conversion file of size 1.5 GB is moved into HDFS. Table 2 shows the time taken to move the files into HDFS in traditional method and the proposed method. Fig 3 shows the chart of time taken by the traditional method and the proposed method to move files into HDFS.

Table 2. Time taken to move files from local file system into HDFS

Technique	File Size in GB	Time in seconds
Traditional	10	398
Proposed	1.5	23

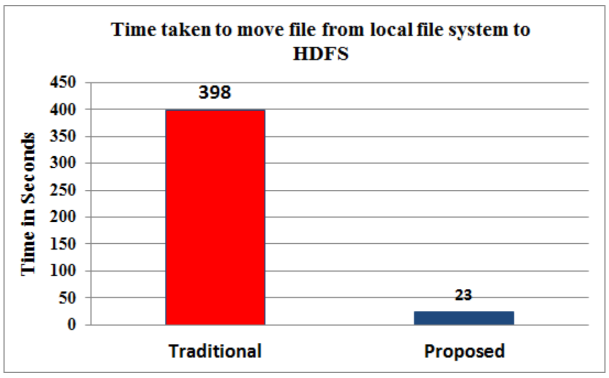

Fig.3. Analysis of time taken by the traditional method and the proposed method to move files into HDFS.

2) Analysis on HDFS Disk space utilization

HDFS disk space utilization analysis is made on both the traditional and the proposed method. In the traditional method, the HDFS disk utilization was 10 GB where we directly moved entire input file of size 10 GB from local file system to HDFS. But in the proposed method we converted the transaction item's name into unique ID, thereby reducing transaction data set size of 10 GB to 1.5 GB, so the HDFS disk utilization was 1.5 GB. Table 3 shows the HDFS disk space utilization by the traditional method and the proposed method. Fig 4 shows the chart of HDFS disk space utilization by the traditional method and the proposed method.

Table 3. HDFS disk space utilization

Technique	HDFS disk space utilization in GB
Traditional	10
Proposed	1.5

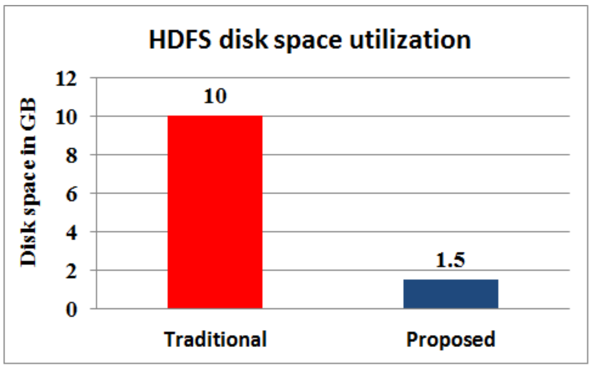

Fig.4. Analysis of time taken by the traditional method and the proposed method to move files into HDFS

3) *Performance analysis of Apriori MapReduce*

Apriori MapReduce applies an iterative approach to find the frequent item set that satisfies the minimum support count threshold. The pseudo code of Apriori MapReduce is shown in Fig.1. HDFS divides large volume of transaction data into data blocks of equal size (i.e. 64 MB). Performance analysis of Apriori MapReduce was made on both the traditional and the proposed method. In the traditional method HDFS disk space utilization by the transaction data set is 10 GB, HDFS divides 10 GB data into 157 data blocks. Time taken to process 157 data blocks from Apriori MapReduce is 72 minutes. In the proposed method HDFS disk space utilization by the transaction data set is 1.5 GB , HDFS divides 1.5 GB data into 24 data blocks. Time taken to process 24 data blocks from Apriori MapReduce is 08 minutes. The proposed method improves the performance of frequent item mining based on Apriori MapReduce by 88.9%.

Table 4 shows Comparison of Apriori MapReduce time taken for analysis by the traditional method and the processed method. Fig 5 shows the time taken by Apriori MapReduce technique in traditional method and the proposed method

Table 4. Comparison of Apriori MapReduce time taken for analysis by the traditional method and the proposed method

Technique	HDFS disk space utilization in GB	Time taken for analysis in Minutes
Traditional	10	72
Proposed	1.5	8

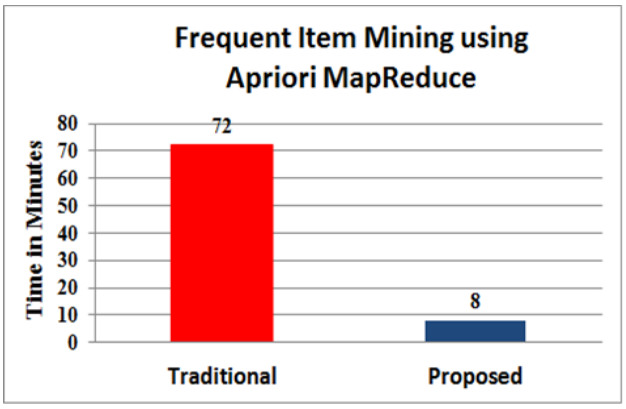

Fig.5. Analysis of Apriori MapReduce by the traditional method and the proposed method

4) *Performance analysis of FP-growth MapReduce*

FP-growth uses novel data structure called frequent pattern tree (FP-tree) to mine frequent pattern. The pseudo code of FP-growth is shown in Fig.1. HDFS divides large volume of transaction data into data blocks of equal size (i.e. 64 MB). Performance analysis of FP-growth MapReduce was made on both the traditional and the proposed method. In the traditional method HDFS disk space utilization by the transaction data set is 10 GB, HDFS divides 10 GB data into 157 data blocks. Time taken to process 157 data blocks from Apriori MapReduce is 59 minutes. In the proposed method HDFS disk space utilization by the transaction data set is 1.5 GB, HDFS divides 1.5 GB data into 24 data blocks. Time taken to process 24 data blocks from Apriori MapReduce is 06 minutes. The proposed method improves the performance of frequent item mining based on FP-growth MapReduce by 89.94%.

Table 5. Comparison of FP-growth MapReduce time taken for analysis by the traditional method and the proposed method

Technique	HDFS disk space utilization in GB	Time taken for analysis in Minutes
Traditional	10	59
Proposed	1.5	06

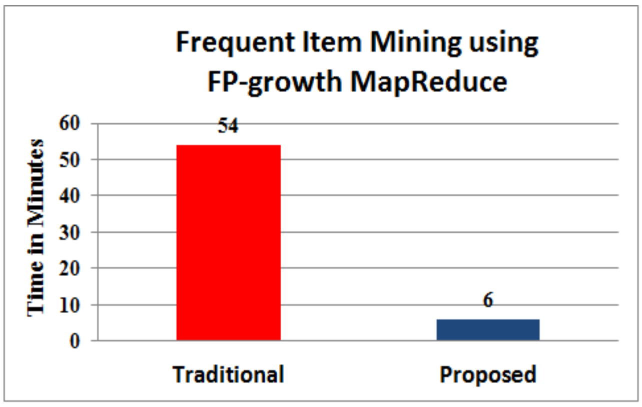

Fig.6. Analysis of FP-growth MapReduce by the traditional method and the proposed method

Table 5 shows comparison of FP-growth MapReduce time taken for analysis by the traditional method and the proposed method. Fig 6 shows the time taken by FP-growth MapReduce technique in traditional method and the proposed method.

VI. Conclusions

Mining frequent item sets in the transaction data can be extremely useful. With exponential growth of data most of the traditional frequent pattern mining algorithms become ineffective to process large data in short time. Hadoop MapReduce framework, a robust distributed computing infrastructure can store, manage and process huge amounts of data in short time. With a comprehensive set of experiments we identified that the parallel implementation of frequent pattern mining algorithm on MapReduce is not optimal in terms of execution time and disk space utilization. In this paper, we proposed a novel approach on Hadoop MapReduce framework to identify frequent item sets. The experimental results shows that the proposed approach on Hadoop MapReduce framework gets better performance in terms of execution time and disk space utilization compared with the baseline Hadoop MapReduce framework.

Acknowledgment

We would like to thank every member of the faculty at SDMIT, MITE and SMVITM for their guidance and support, which has helped us, complete this research project successfully.

Reference

[1] Hui Chen , Tsau Young Lin, Zhibing Zhang and Jie Zhong (2013) "Parallel Mining Frequent Patterns over Big Transactional Data in Extended MapReduce ", 2013 IEEE International Conference on Granular Computing (GrC), pp.43-48.

[2] Zahra Farzanyar, Nick Cercone(2013) " Efficient Mining of Frequent itemsets in Social Network Data based on MapReduce Framework", 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining,pp.1183-1188.

[3] Yanfeng Zhang, Shimin Chen, Qiang Wang, and Ge Yu (2015) "i2 MapReduce: Incremental MapReduce for Mining Evolving Big Data", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 27,NO. 7,pp.1906-1919.

[4] LI Bing and LI Bing (2014) "A Paralleled Big Data Algorithm with MapReduce Framework for Mining Twitter Data", 2014 IEEE Fourth International Conference on Big Data and Cloud Computing,pp.121-128.

[5] Sheela Gole and Bharat Tidke (2015) "Frequent Itemset Mining for Big Data in social media using ClustBigFIM algorithm", 2015 IEEEInternational Conference on Pervasive Computing (ICPC), pp.1-6.

[6] Yen-hui Liang and Shiow-yang Wu (2015) " Sequence-Growth : A Scalable and Effective Frequent Itemset Mining Algorithm for Big Data Based on MapReduce Framework", 2015 IEEE International Congress on Big Data, pp.393-400.

[7] Zhuobo Rong, DawenXia and Zili Zhang (2013) "Complex Statistical Analysis of Big Data: Implementation and Application of Apriori and FP Growth Algorithm Based on MapReduce". 2013 IEEE conference, pp.968-972.

[8] Pekka Paakkonen and Daniel Pakkala (2015) "Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems", Elsevier Big Data Research, pp.166-186.

[9] Arantxa Duque Barrachina and Aisling O'Driscoll (2014) "A big data methodology for categorizing technical support requests using Hadoop and Mahout", Journal of Big Data, pp. 1-11.

[10] Matthew Herland, Taghi M Khoshgoftaar and Randall Wald (2014) "A review of data mining using big data in health informatics", Journal of Big Data, pp. 1-35.

[11] Dilpreet Singh and Chandan K Reddy (2014) "A survey on platforms for big data analytics", Journal of Big Data, pp. 1-20.

[12] J. Dean and S. Ghemawat, (2004) "Mapreduce: Simplified data processing on large clusters," in Proc. 6th Conf. Symp. Opear. Syst. Des.Implementation, p. 10-18.

[13] A. Labrinidis and H. V. Jagadish (2012) "Challenges and opportunities with big data," in Proceedings of the VLDB Endowment. VLDB, pp. 2032–2033.

[14] J. Han, J. Pei, and Y. Yin (2007) "Mining frequent patterns without candidate generation," in Proceedings of the 9th international conference on Parallel Computing Technologies, pp. 623–631.

[15] R. Agrawal and R. Srikant (1994) "Fast algorithms for mining association rules in large databases," in Proceedings of 20th International Conference on Very Large Data Bases, pp. 487–499.

[16] J. H. Chang and W. S. Lee (2003) "Finding recent frequent itemsets adaptively over online data streams," in Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 487–492.

[17] Le Zhou; Zhiyong Zhong; Jin Chang; Junjie Li; Huang, J.Z,Shengzhong Feng, "Balanced parallel FP-Growth with MapReduce," Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on , pp.28-30

[18] Li N., Zeng L., He Q. & Shi Z. (2012). Parallel Implementation of Apriori Algorithm Based on MapReduce. Proc. of the 13 ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD'12). Kyoto, IEEE: 236 – 241.

[19] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh (2012) "Apriori-based frequent itemset mining algorithms on MapReduce". In Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12). ACM, New York, pp 86-93

Authors' Profiles

Dr. Nagesh H.R, Dean(Academic), Professor & Head, Department of Computer Science & Engineering, Mangalore Institute of Technology & Engineering, Moodbidri, has got his M.Tech and Ph.D(Computer Engineering) from NITK Surathkal. He has published more than 50 research papers in National and International Conferences and journals. He has delivered more than 20 invited talks in topics like 'Component Based Software Development', 'Internet Security', 'Web Security', 'Web Engineering', 'Information Security' ,'Network Management', 'Promoting Global Cyber Security' ,'Security issues in Distributed Systems', 'Digital library and Information Search', 'Information Security Management' ,'Recent Trends in Information Technology' and 'Security issues in Cloud Computing'. He has also chaired many sessions in International and National level technical paper presentations. He has also published one chapter titled 'Proactive models for Mitigating Internet DoS/DDoS Attacks', in 'Selected Topics in Communication Networks and Distributed Systems', World Scientific, London, April 2010. He had also worked as Visiting faculty to NITK Surathkal and NITK-Science and Technology Entrepreneurs Park, Karnataka, Surathkal. Published two books titled 'Fundamentals of CMOS VLSI Design' for V semester Electronics & Communication Engineering students of VTU : Pearson Education & 'VLSI Design' for V semester Electronics & Communication Engineering students of JNTU : Pearson Education. Member of BOS for PG studies in Computer Science at Mangalore University and Manipal Institute of Technology for PG studies in Computer Science & Engineering. Worked as member of BOE and Exam coordinator in VTU Belgaum. Member of BOS in Computer Science & Engineering of VTU Belgaum for year 2013 to 2016.

Mr. Guru Prasad M S, Asst .professor, Dept of Computer Science & Engineering. Shri Dharmasthala Manjunatheshwara Institute of Technology, Ujire, Dakshinna Kannada. He got his M.Tech (Computer Engineering) from NMAMIT Nitte. He has published 6 research papers in International Conferences and Journals. He has delivered 09 invited talks on "Big Data Analytics".

Ms. Swathi Prabhu, Asst. Professor, Dept. of Computer Science & Engineering, Shri Madhwa Vadiraja Institute of Technology & Management, Bantakal, Udupi. She got her M.Tech (Computer Engineering) from NMAMIT Nitte. She has published few research papers in National and International Conferences and journals. Her interested area is BigData Analysis using Hadoop, Distributed computing, parallel computing.

How to cite this paper: Guru Prasad M S, Nagesh H R, Swathi Prabhu,"High Performance Computation of Big Data: Performance Optimization Approach towards a Parallel Frequent Item Set Mining Algorithm for Transaction Data based on Hadoop MapReduce Framework", International Journal of Intelligent Systems and Applications(IJISA), Vol.9, No.1, pp.75-84, 2017. DOI: 10.5815/ijisa.2017.01.08